REPORT

## Data:

The dataset is taken from UCI Repository named as **Heterogeneity Activity Recognition Data Set**. The data is characterized as Multivariate and is used for associated task like Clustering and classification.

The dataset contains the readings of two motion sensors commonly found in smartphones. Reading were recorded while users executed activities scripted in no specific order carrying smartwatches and smartphones.

Total Activities: ‘Biking’, ‘Sitting’, ‘Standing’, ‘Walking’, ‘Stair Up’ and ‘Stair down’.

(1) , (2), (3), (4), (5), (6)

Total Sensors: Two embedded sensors, i.e., Accelerometer and Gyroscope, sampled at the highest frequency the respective device allows.

Total Devices: 4 smartwatches (2 LG watches, 2 Samsung Galaxy Gears)

8 smartphones (2 Samsung Galaxy S3 mini, 2 Samsung Galaxy S3, 2 LG Nexus 4, 2 Samsung Galaxy S+)

Used Devices: (5) nexus\_2b, nexus\_1c, nexus\_2c, s3\_1, s3\_2, s3\_mini\_1

Total Recordings: 9 users data recordings

Used Recordings: only 3 users data we considered

## Filtering data:

We got the data with 128 features which is not required for the present classification so firstly for all the below models we filtered the data such that it contains only (4 parameters) sensor outputs and its corresponding output label, and instead of data with char labels we modify the labels with integers such it’s flexible for us to do operations faster. We wrote this code in R programming and submitted in the zip file.

## Difficulty of the proposed work:

1. Dealing with Huge data of about 3.07 GB.
2. Real time data with little variation in coordinates to identify exactly into which category the test data falls in.
3. Dealing with various devices where there is possibility in variation of X,Y and Z coordinates data collection like the rate at which different devices recognize the variations in actions.

Algorithms:

The clustering and classifications strategies used for the above dataset in order to identify and classify the action performed by the user are:

1. K nearest neighbour.
2. Logistic Regression (one vs all)
3. XGBoost
4. Random Forest

K Nearest Neighbors:

1. **Description:** Based on the majority label of the k nearest neighbour in the training data we are predicting the output for all the test instance.

1. **Complexity of the algorithm:** The data which we have to classify is having 3 input features to of each coordinate value and its label, by using the same parameters we trained the data (70% of input) and tested with rest (30% of input) when we changed the parameters by considering the time also along with 3 coordinate parameters the results are falling from 85% to 62% so we came to know that the time parameters has not much importance because it makes sense because the time has to nothing to do with predicting the action so we trained and tested with 3 features with 1 column as label and results came as shown below:

3) **Implementation Part**: Implementation part we completely done from the scratch and we

used the parallel programming with 10 processor’s in order to solve this because this the

algorithm which is taking huge time when compared to any other above mentioned

algorithms but the accuracy is pretty good when compared to other.

Logistic Regression:

1. **Description:** that is usually taken to apply to a binary dependent variable.A logistic model is one where the probability of an event is a linear combination of independent or predictor variables. In this problem setting we are applying Logistic Regression Model in a multiclass setting (using one vs all method) in order to predict what action the user’s is most probably going to take as recorded by a specific device by making use of the sensors.

2.) **Complexity of the Data:**

The data which we have to classify is having 3 input features to of each coordinate value and its label, by using the same parameters we trained the data (70% of input) and tested with rest (30% of input) when we changed the parameters by considering the time also along with 3 coordinate parameters the results are falling from 83.33% to 60% so we came to know that the time parameters has not much importance because it makes sense because the time has to nothing to do with predicting the action so we trained and tested with 3 features with 1 column as label and results are shown in the figure below:

3.) **Implementation Part**:

As discussed above in order to classify the Logistic regression model in the multivariate settings we are required to make use of **one vs all method .** So as there are 6 different actions a user can perform like ‘Biking’, ‘Sitting’, ‘Standing’, ‘Walking’, ‘Stair Up’ and ‘Stair down’. By training the model on one vs all setting basically so considering the whole problem setting to be like two binary classes 1st one of the classification label considered as a class and 2nd all the other labels is considered another class.

4.) **References:**

http://mlwiki.org/index.php/One-vs-All\_Classification

XGBoost:

1. **Description:** The reason why we used this is XGBoost algorithm because its a library designed and optimized for boosted tree algorithms which is very useful and opt to deal with current huge data we are having. As expected it’s giving the results faster than any other above mentioned algorithms.
2. **Complexity of the Data:**The data which we have to classify is having 3 input features to of each coordinate value and its label, by using the same parameters we trained the data (70% of input) and tested with rest (30% of input) when we changed the parameters by considering the time also along with 3 coordinate parameters the results are falling from 85% to 62% so we came to know that the time parameters has not much importance because it makes sense because the time has to nothing to do with predicting the action so we trained and tested with 3 features with 1 column as label and results came as shown below:
3. **Implementation:** We didn’t implemented this algorithm from scratch. we used the inbuilt libraries to trained and tested with above mentioned data and as mentioned above the execution is really fast and accuracy is mostly near to any other above mentioned algorithms.
4. **Reference’s:** https://github.com/rasbt/mlxtend/issues/170

**Random Forest:**

1. **Description:**

**Random forests** or **random decision forests** are the method for classification, regression and other task that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

2.) **Complexity of the Data:**

The data which we have to classify is having 3 input features to of each coordinate value and its label, by using the same parameters we trained the data (70% of input) and tested with rest (30% of input) when we changed the parameters by considering the time also along with 3 coordinate parameters the results are falling from 83.33% to 60% so we came to know that the time parameters has not much importance because it makes sense because the time has to nothing to do with predicting the action so we trained and tested with 3 features with 1 column as label and results are shown in the figure below:

3.)

**Implementation:**

We didn’t implemented this algorithm from scratch. we used the inbuilt libraries to trained and tested with above mentioned data and as mentioned above the execution is really fast and accuracy is mostly near to any other above mentioned algorithms.

4.)

**References:**

http://www.blopig.com/blog/2017/07/using-random-forests-in-python-with-scikit-learn/

Comparisons of various Algorithms:

Algorithms Time(avg) Complexity(avg)

K-Nearest-neighbors

Logistic-Regression

XGBoost

Rand\_forest

## Important References:

Relevant Link:

http://archive.ics.uci.edu/ml/machine-learning-databases/00344/

Relevant Papers:

Smart Devices are Different: Assessing and Mitigating Mobile Sensing Heterogeneities for Activity Recognition

**Filter\_nexus1\_b\_new.csv' - Accuracy: 79.45%**

**Filter\_nexus2\_a\_new.csv 86.17%**

**Filter\_nexus2\_b\_new - 77.43%**

**Filter\_nexus\_1\_c\_new - 82.82**

**Filter\_nexus\_2\_c\_new - 83.69**

**Filter\_s3\_1\_a\_new - 85.84**

**Filter\_s3\_1\_b\_new - 80.95%**

**Filter\_s3\_1\_c\_new- 83.59**

**Filter\_s3\_2\_a\_new- 86.09**

**Filter\_s3\_2\_b\_new- 81.41**

**Filter\_s3\_2\_c\_new - 83.73**

**Filter\_s3mini1\_1\_a\_new- 85.85**

**Filter\_s3mini1\_1\_b\_new- 81.31**

**Filter\_s3mini1\_1\_c\_new- 83.08**

**Random Forest**

**Filter\_nexus2\_b\_new -**

**Train Accuracy :: 98.79568399501095**

**Test Accuracy :: 79.78475139335035**

**Filter\_nexus\_2\_a\_new-**

**Train Accuracy :: 99.08250918672385**

**Test Accuracy :: 86.93464565843706**

**Filter\_nexus\_2\_b\_new-**

**Train Accuracy :: 98.69503628111357**

**Test Accuracy :: 77.86816685553937**

**Filter\_nexus\_1\_c\_new-**

**Train Accuracy :: 99.07510796012538**

**Test Accuracy :: 83.38983582886021**

**Filter\_nexus\_2\_c\_new:**

**Train Accuracy :: 99.04104546291184**

**Test Accuracy :: 84.1354825779483**

**Filter\_s3\_1\_a\_new:**

**Train Accuracy :: 98.99379642572461**

**Test Accuracy :: 85.94571282244804**

**Filter\_s3\_1\_b\_new:**

**Train Accuracy :: 98.86135069716701**

**Test Accuracy :: 79.96765127008538**

**Filter\_s3\_1\_c\_new:**

**Train Accuracy :: 98.86135069716701**

**Test Accuracy :: 79.96765127008538**

**Filter\_s3\_2\_a\_new:**

**Train Accuracy :: 99.05527638190955**

**Test Accuracy :: 86.27748503324413**

**Filter\_s3\_2\_b\_new:**

**Train Accuracy :: 98.79250079440737**

**Test Accuracy :: 80.93207996822454**

**Filter\_s3\_2\_c\_new:**

**Train Accuracy :: 98.97085795801915**

**Test Accuracy :: 83.45815501664289**

**Filter\_s3min1\_1\_a\_new:**

**Train Accuracy :: 99.11279196993483**

**Test Accuracy :: 87.53191253191254**

**Filter\_s3min1\_1\_b\_new:**

**Train Accuracy :: 99.01016157824873**

**Test Accuracy :: 82.60486476383734**

**Filter\_s3min1\_1\_c\_new:**

**Train Accuracy :: 99.06145281230391**

**Test Accuracy :: 84.61466575291259**

**Logistic regression:**

**['nexus1\_user\_c','nexus2\_userb,','nexus2\_userc','s3\_usera','s3\_userb','s3\_userc','s3\_2\_usera','s3\_2\_userb','s3\_2\_userc','s3mini1\_usera','s3mini1\_userb','s3mini1\_userc'], [83.33,82.5,83.319,84.135,83.33333333333333,83.13,84.54,83.33,84.458,83.31,82.604,84.0]**

**Filter\_nexus2\_b\_new - 83.33**

**Filter\_nexus\_1\_c\_new - 83.31763844202045**

**Filter\_nexus\_2\_c\_new - 83.31901776085358**

**Filter\_s3\_1\_a\_new - 83.33333333333333**

**Filter\_s3\_1\_b\_new - 83.33333333333333**

**Filter\_s3\_1\_c\_new- 83.3294329871776**

**Filter\_s3\_2\_a\_new-** 83.33333333333333

**Filter\_s3\_2\_b\_new- 83.31038439472174**

**Filter\_s3\_2\_c\_new - 83.28578362903305**

**Filter\_s3mini1\_1\_a\_new- 83.32963343230568**

**Filter\_s3mini1\_1\_b\_new- 83.33333333333333**

**Filter\_s3mini1\_1\_c\_new- 83.27726206051544**

**K-nearest neighbours:**

**Filter\_nexus2\_b\_new - Accuracy 0.785000 Time 216.855949 secs.**

**Filter\_nexus\_1\_c\_new - Accuracy 0.800000 Time 937.536304 secs.**

**Filter\_nexus\_2\_c\_new - Accuracy 0.811000 Time 949.315085 secs.**

**Filter\_s3\_1\_a\_new - Accuracy 0.837000 Time 771.462868 secs.**

**Filter\_s3\_1\_b\_new - Accuracy 0.772000 Time 769.791426 secs.**

**Filter\_s3\_1\_c\_new- Accuracy 0.800000 Time 740.191573 secs**

**Filter\_s3\_2\_a\_new-** Accuracy 0.822000 Time 745.721201 secs.

**Filter\_s3\_2\_b\_new- Accuracy 0.783000 Time 761.146303 secs.**

**Filter\_s3\_2\_c\_new - Accuracy 0.825000 Time 98.554915 secs.**

**Filter\_s3mini1\_1\_a\_new- Accuracy 0.852000 Time 204.651255 secs.**

**Filter\_s3mini1\_1\_b\_new- Accuracy 0.828000 Time 206.627811 secs**

**Filter\_s3mini1\_1\_c\_new- Accuracy 0.812000 Time 93.527510 secs.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **K nearest neighbour** | **Logistic Regression** | **XG Boost** | **Random Forest** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |